SALVATORE DI GIROLAMO <DIGIROLS@INF.ETHZ.CH>

DPHPC: Sequential Consistency

Recitation session

Systems@ETH zürich

Consistency vs Coherence

- **Writes to same location**
  - **Coherence**
    - a) ***Write Serialization:*** *all processors see writes to the same location in the same order*
    - b) ***Write Propagation:*** *a write will eventually be seen by other processors*

- **Writes to different location**
  - **Memory Model:** defines the ordering of writes and reads to different memory locations – the hardware guarantees a certain consistency model and the programmer attempts to write correct programs with those assumptions

P1	P2
Y=10	while (X==0)
X=2	Z=Y

Consistency: Example

- **Multiprocessor with bus-based snooping cache-coherence and write buffer**
- **Initially A=B=0**

```
P1:
A=1
if (B==0){
   <enter critical section>
}
```

```
P2:
B=1
if (A==0){
   <enter critical section>
}
```

Does it work?

- This lock implementation is based on two different variables (i.e., memory location)
- The stores are intercepted by the write buffer => P1 and P2 can enter the critical section at the same time
- Cache coherence is not involved here

Are we sure?

Consistency: Example

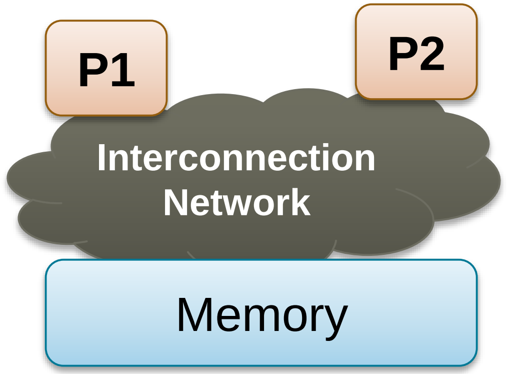
- **Multiprocessor without cache**
- **Initially A=B=0**

```
P1:
A=1
if (B==0){
   <enter critical section>
}
```

```
P2:
B=1
if (A==0){
   <enter critical section>
}
```

Does it work?

Updates take time to propagate!



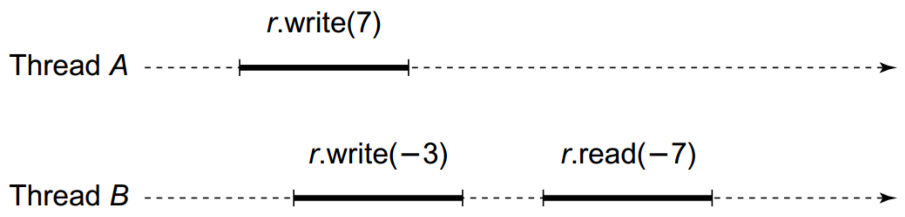
Memory Models

"A formal specification of how the memory system will appear to the programmer, eliminating the gap between the behavior expected by the programmer and the actual behavior supported by a system." [Adve' 1995]

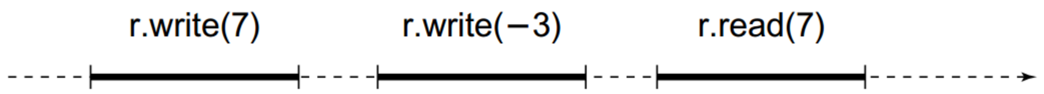
- **Memory model specifies:**
  - How threads interact through memory
  - What value a read can return
  - When does a value update become visible to other threads
  - What assumptions are allowed to make about memory when writing a program or applying some program optimization

Sequential Consistency

- **Method calls act as if they occurred in a sequential order consistent with program order**
  - ***Method calls should appear to happen in a one-at-time, sequential order***



  - ***Method calls should appear to take effect in program order***

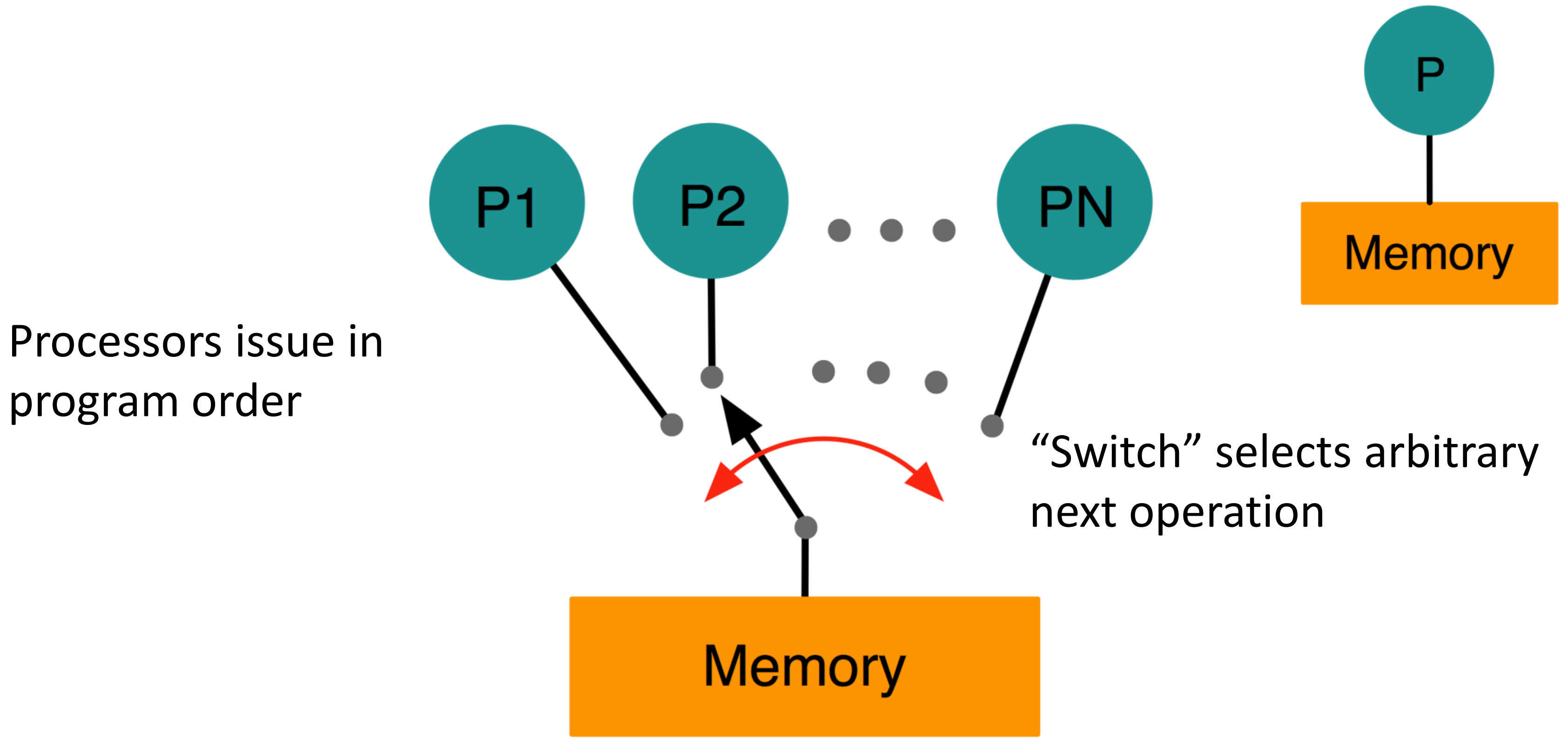


Program Order: Per-processor order of memory accesses, determined by program‘s *control flow*.

Visibility Order: Order of memory accesses observed by one or more processors

Sequential Consistency Illustrated

- **Method calls act as if they occurred in a sequential order consistent with program order**
  - ***Method calls should appear to happen in a one-at-time, sequential order***
  - ***Method calls should appear to take effect in program order***


P1
P2
PN
P
Memory
Processors issue in program order
"Switch" selects arbitrary next operation
Memory

Sequential Consistency - Discussion

- **Programmer's view:**
  - Prefer sequential consistency
  - Easiest to reason about

- **Compiler/hardware designer's view:**
  - Sequential consistency disallows many optimizations!
  - Substantial speed difference
  - ➢ Most architectures and compilers don't adhere to sequential consistency!

- **Solution: synchronized programming**
  - Access to shared data (aka. "racing accesses") are ordered by synchronization operations
  - Synchronization operations guarantee memory ordering (aka. fence)
  - More later!

Memory Fence: special instructions that require all previous memory accesses to complete before proceeding *(sequential consistency)*

Relaxed Memory Models

- **Ideal: intuitive programming model (i.e., sequential consistency) and high-performance**
  - *Not that easy* ☺

- **Idea: Relax some constraints, but allow the programmer to enforce them from specific portions of the code**

- **Some possible relaxations (different memory locations):**
  - Relax W→R: *Reads may be reordered with older writes to different locations but not with older writes to the same location* (x86)
  - Relax W→W: *Writes can be reordered with other writes*
  - Relax R→W: *Writes can be reordered with older reads*

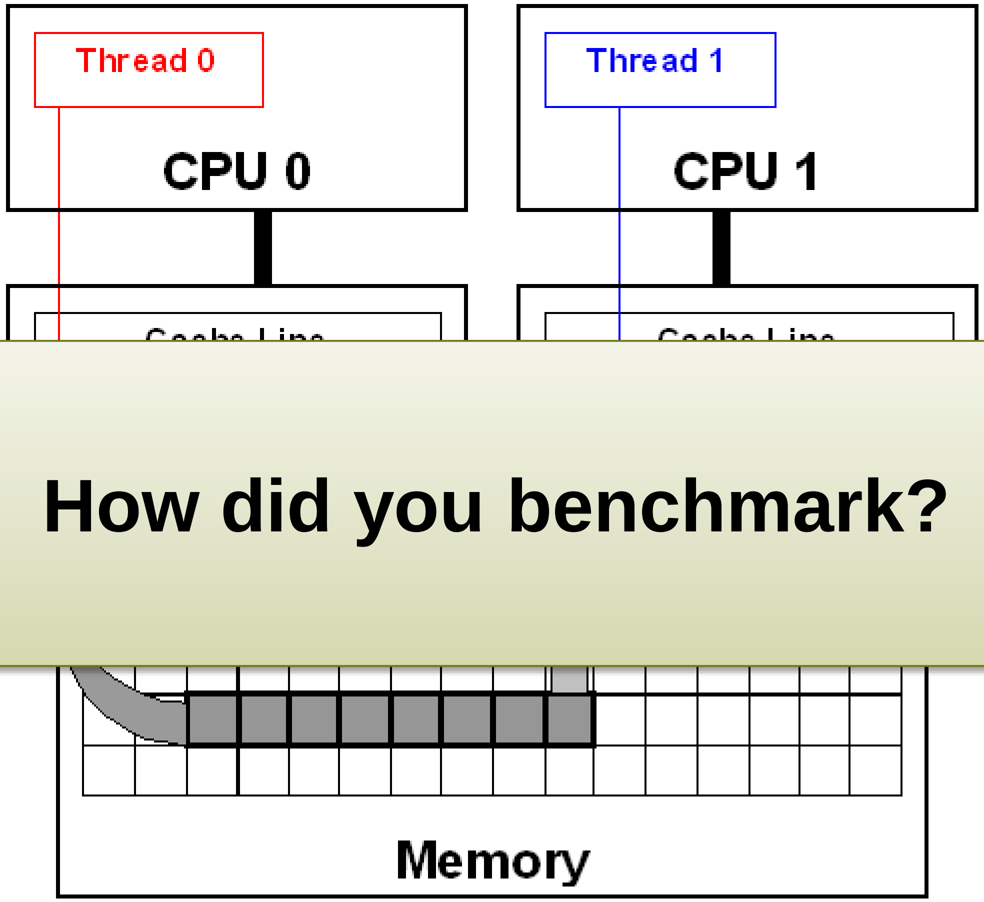
- **A consistency model is identified by a set of contraint**

SALVATORE DI GIROLAMO <DIGIROLS@INF.ETHZ.CH>

DPHPC: Assignment Discussion

Recitation session

False Sharing Benchmark



How did you benchmark?

https://software.intel.com/sites/default/files/m/d/4/1/d/8/5-4-figure-1.gif

False Sharing Benchmark

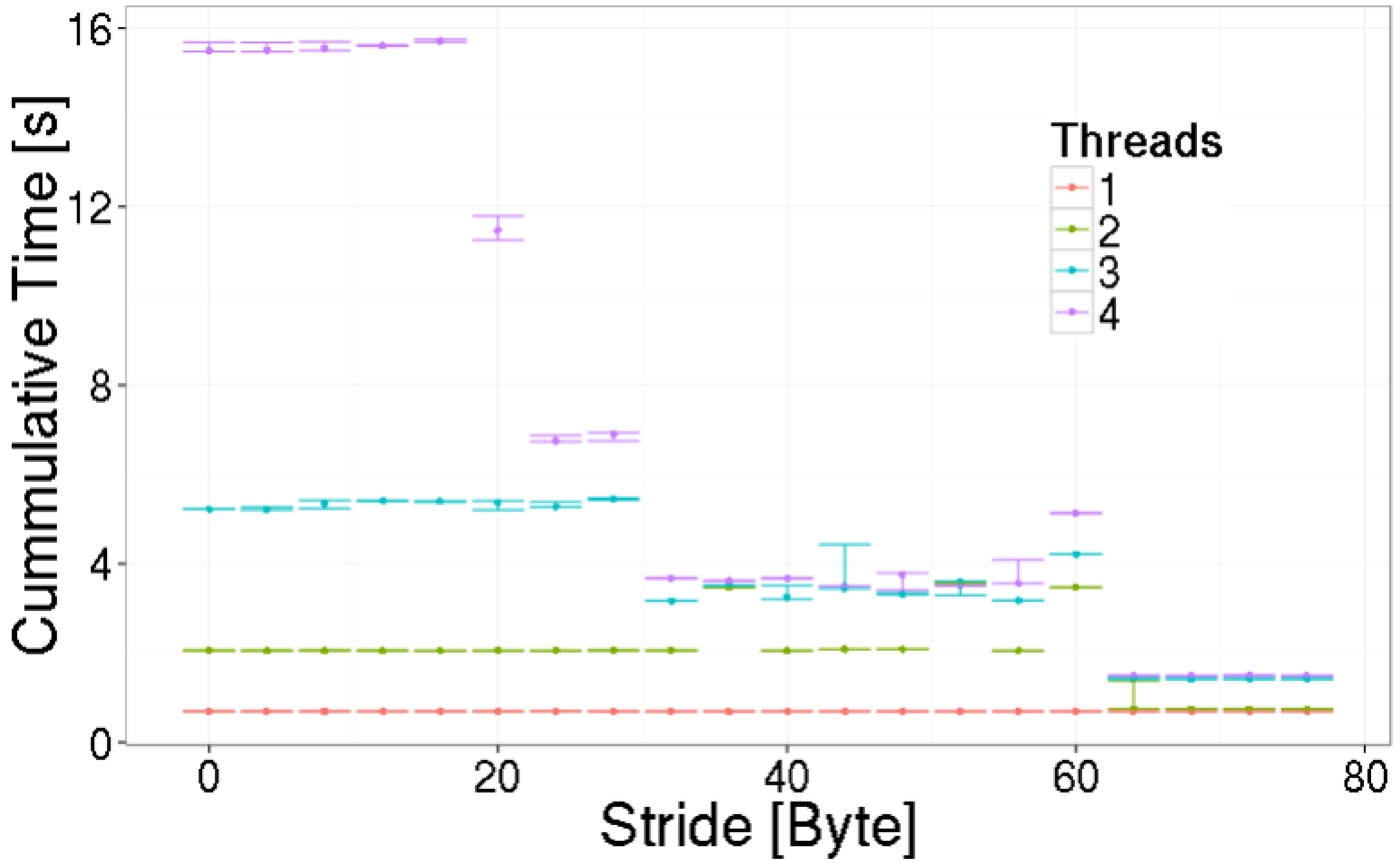
- **Idea: Allocate uint8_t array a, let core 0 write to a[0] and core 1 to a[x]**

- **If x is larger than the size of one CL this should be "fast" because both cores operate on their on cached copy of different CLs**

- **If x is smaller than one CL it will be slow, due to false sharing**

- **In practice it is a bit harder to get it right :)**
  - If we write only once it might not really be parallel -> do it in a large enough loop
  - If we write only one Byte in each iteration we will not see much because of loop overhead (incrementing counter, jump) -> write 8 bytes in inner loop
  - Make sure the compiler does not "optimize" your loop by removing it!

False Sharing Benchmark



Machine: Intel Core i5 3230M; **Compiler:** gcc 4.9.1 –O3 –fopenmp –std=gnu11

On Benchmarking / Plots

- **Make sure you can explain your data!**
- **Plots should**
  - have labels + units on x and y axis
  - have legends or a description of each line/color
  - some indication of accuracy of measurements
  - do not measure only once or show only the minimum!
- **More details on this topic will follow!**

- **You can make plots with many different software packages**
- **We (SPCL) usually use GNU R**
  - Free Software
  - Includes many statistic / data-analysis functions
  - Probably harder to learn than Excel/Gnuplot, but generates nicer plots ☺

SALVATORE DI GIROLAMO <DIGIROLS@INF.ETHZ.CH>

DPHPC: OpenMP - Synchronization

Recitation session

Based on http://openmp.org/mp-documents/Intro_To_OpenMP_Mattson.pdf

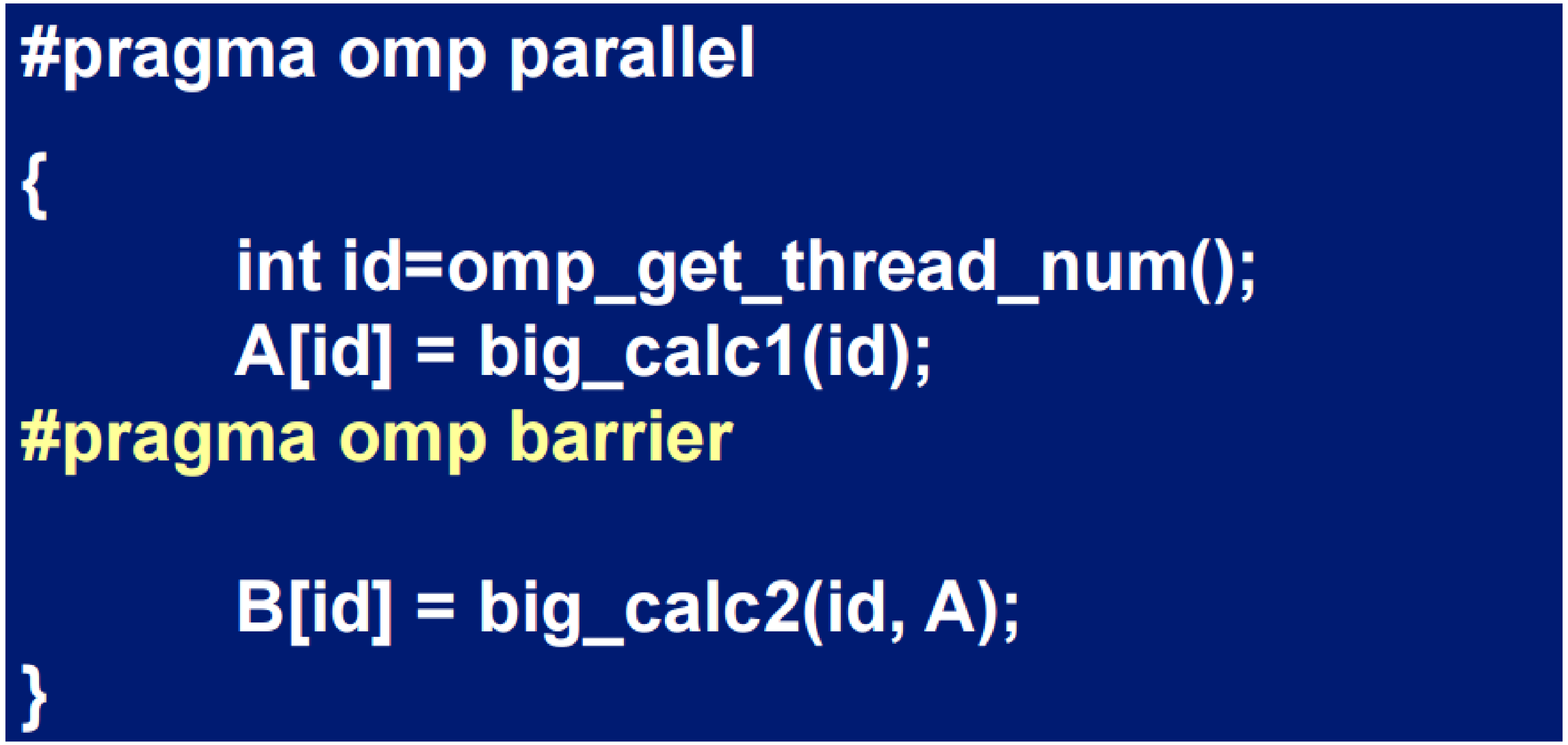
OpenMP - Synchronization

Synchronization is used to impose order constraints and to protect access to shared data

- **High level synchronization:**
  - Critical, Atomic, Barrier, Ordered
- **Low level synchronization**
  - Flush, Locks (both simple and nested)

Barrier

- **Each thread waits until all threads arrive.**



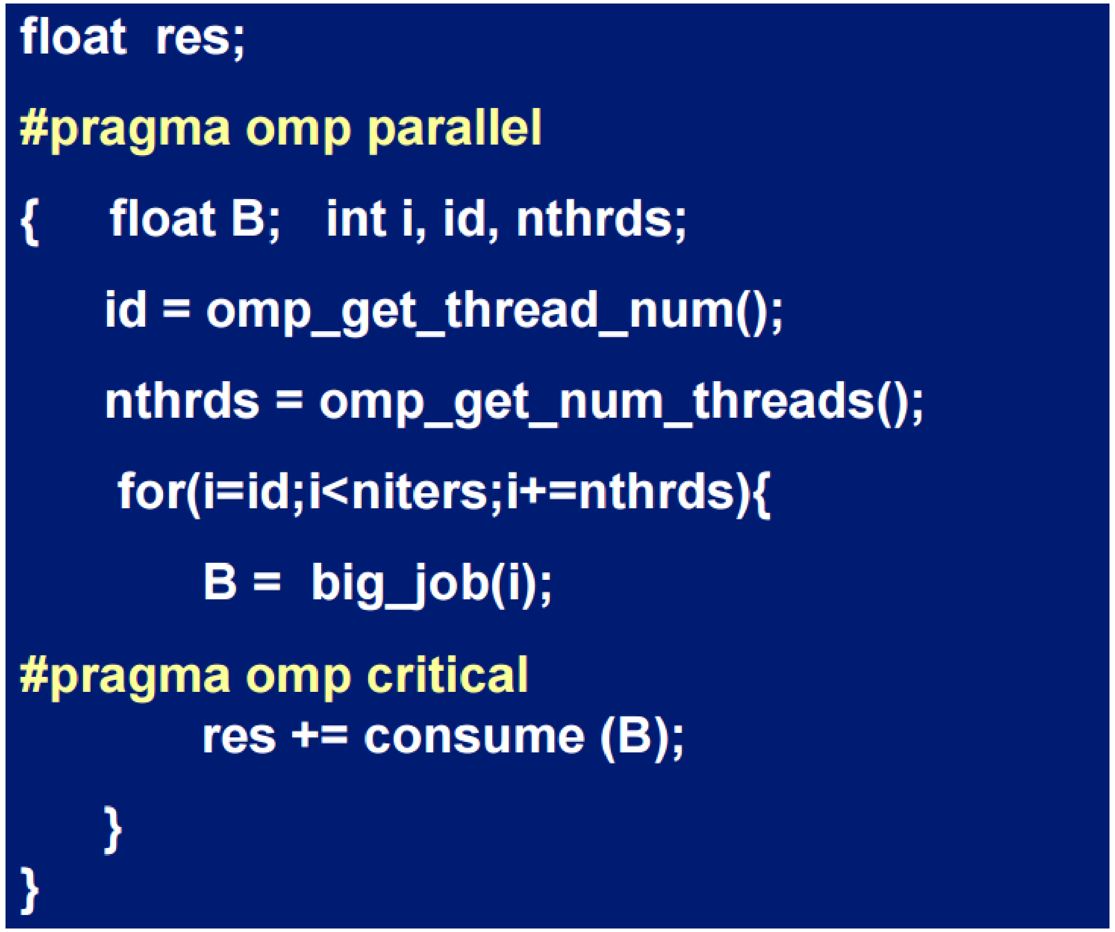
```
#pragma omp parallel

{
        int id=omp_get_thread_num();
        A[id] = big_calc1(id);
#pragma omp barrier


        B[id] = big_calc2(id, A);
}
```

Critical

- **Mutual exclusion: Only one thread at a time can enter a critical region**



```
float  res;
#pragma omp parallel
{     float B;   int i, id, nthrds;
      id = omp_get_thread_num();
      nthrds = omp_get_num_threads();
       for(i=id;i<niters;i+=nthrds){
             B =  big_job(i);
#pragma omp critical
             res += consume (B);
      }
}
```

Atomic

- **Atomic provides mutual exclusion but only applies to the update of a memory location (the update of X in the following example)**

```
#pragma omp parallel
{
        double tmp, B;
       B =  DOIT();
       tmp = big_ugly(B);
 #pragma omp atomic
        X +=  tmp;
}
```

The statement inside the atomic must be one of the following forms:

- x binop= expr
- x++
- ++x
- x—
- --x

X is an lvalue of scalar type and binop is a non-overloaded built in operator.

Event: LLVM Compiler and Code Generation Social

When: ***13.10.2016 19:00***

Where: ***ETH Zurich, CAB, E72***

The LLVM Compiler and Code Generation Social is a meetup to discuss compilation and code generation topics, with a special focus on LLVM, clang, Polly, and related projects. If you are interested in generating code for a variety of architectures, (static) program analyses for real world C/C++/OpenCL/CUDA programs, building your own programming language, register allocation and instruction selection, software hardening techniques, have an idea for a great optimization, or want to target GPUs and FPGA, ... This event is for you!

Our primary focus are free discussions between interested people (+ beer and food). This is a great opportunity to get and discuss project ideas or to just learn about what people at ETH and around Zurich are doing.

Contact: Tobias Grosser *(https://www.inf.ethz.ch/personal/tgrosser/)*